ON THE ANALYSIS OF CMMN EXPRESSIVENESS: REVISITING WORKFLOW PATTERNS

Renata Carvalho [1], *Anis Boubaker* [1], *Javier Gonzalez-Huerta* [1], *Hafedh Mili* [1], *Simon Ringuette* [2], *and Yasmine Charif* [3]

Mars 2016

Département d'informatique

Université du Québec à Montréal

Rapport de recherche Latece 2016-1

Laboratoire de recherche sur les technologies du commerce électronique

ON THE ANALYSIS OF CMMN EXPRESSIVENESS: REVISITING WORKFLOW PATTERNS

Renata Carvalho [1], *Anis Boubaker* [1], *Javier Gonzalez-Huerta* [1], *Hafedh Mili* [1], *Simon Ringuette* [2], *and Yasmine Charif* [3]
[1] *Département d'informatique*
UQAM
Montréal, Qc, Canada
[2] *Trisotech Inc.*
Montréal, Qc, Canada
[3] *Xerox Innovation Group*
Xerox Research Center Webster
Mailstop 128-29E

Laboratoire de recherche sur les technologies du commerce électronique
Département d'informatique
Université du Québec à Montréal
C.P. 8888, Succ. Centre-Ville
Montréal, QC, Canada
H3C 3P8
`http://www.latece.uqam.ca`

Mars 2016

Rapport de recherche Latece 2016-1

Summary

Traditional business process modeling languages use an *imperative style* to specify all *possible execution flows*, leaving little flexibility to process operators. Such languages are appropriate for low-complexity, high-volume, mostly automated processes. However, they are inadequate for *case management*, which involves low-volume, high-complexity, knowledge-intensive work processes of today's knowledge workers. OMG's *Case Management Model and Notation*(CMMN), which uses a *declarative stytle* to specify *constraints* placed at a process execution, aims at addressing this need. To the extent that typical case management situations do include at least some measure of imperative control, it is legitimate to ask whether an analyst *working exclusively in CMMN* can *comfortably* model the range of behaviors s/he is likely to encounter. This paper aims at answering this question by trying to express the extensive collection of Workflow Patterns in CMMN. Unsurprisingly, our study shows that the workflow patterns fall into three categories: i) the ones that are handled by CMMN basic constructs, ii) those that rely on CMMN's engine capabilities and iii) the ones that cannot be handled by current CMMN specification. A CMMN tool builder can propose patterns of the second category as companion modeling idioms, which can be translated behind the scenes into standard CMMN. The third category is problematic, however, since its support in CMMN tools will break model interoperability.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Organisations achieve their missions by executing domain-specific core business processes, i.e. ordered sequences of activities that bring value to their customers/users [6]. Business Process Management (BPM) denotes a set of practices and tools that aim at, 1) proper modeling of an organization's business processes, 2) an analysis of those process models to identify opportunities for elimination or automation, and 3) the actual execution and monitoring of thus simplified/optimized processes [16, 10]. Skeptics would argue that early generation BPM and BPM systems (BPMS) have only solved the 'easy problems' [18]: 1) business processes where a handful of contingencies handle all situations, and 2) processes where most of the activities can be automated. However, knowledge intensive processes do not lend themselves to such rigid formalizations: both the set of activities to perform, and the content of each activity, depend heavily on a combination of domain-specific knowledge, and the case at hand: that is case management [11]. Anyone who has been to an emergency room, or has watched (rerun) episodes of the wildly successful ER television series, can attest to the loosely 'organized' but often suboptimal chaos that reigns in emergency rooms. Such processes would greatly benefit from BPM technology through more information sharing, a better quality of service, and a better use of scarce resources. However, they are also complex to formalize in prescriptive process modeling languages such as BPMN or EPC. Indeed, hospital staff have somewhat well-defined protocols to follow, but are otherwise left to contend with, 1) mostly manual activities, including physical tasks (e.g. taking a blood sample) and knowledge intensive tasks (e.g. diagnosis), 2) different instances of the same process that compete for scarce resources, and 3) the urgency to act.

The Case Management Model and Notation, a recent OMG standard, has been created expressly to handle such processes. CMMN uses a *declarative* process specification style where the focus is *not* on representing the *possible execution sequences* of the process (the *closed world* style), but rather on specifying the *constraints* that apply to the process (the *open world* style), leaving the choice of the tasks to execute, and the sequence between them, to the human operator/case manager–as long as those *constraints are respected*. The emergence of *declarative* process modeling languages– perhaps culminating in the emergence of an "industrial standard"–raises

the age-old debate between the merits of the declarative versus imperative style that has permeated a number of computer science realms, from formal specification languages to programming languages. While the theoretical issues concerning the relative expressive power of the languages and the class of problems or behaviors that can be specified in either family/approach are interesting in their own right, we are interested in the more pragmatic issues of *modeling convenience*. Indeed, to the extent that, 1) a modeling language is (or should be) designed primarily for human *production* (business/process analysts) and *consumption*, *and* 2) most processes, including the ones typical of case management, include at least some measure of prescriptive/imperative control, it is legitimate to ask whether an analyst *working exclusively in CMMN* can *comfortably* model the range of behaviors s/he is likely to encounter in case management[1]. One way to answer this question is to explore the extent to which CMMN can handle common behaviors found in typical workflow systems. To this end, we use the catalog of behaviors inventoried in the *workflow patterns initiative*. Indeed, such an initiative developed a collection of well-known, widely applied workflow best practices from different perspectives (control flow, data, resource, and exception handling), independently from an implementation technology. The ability to represent such patterns (or a subset thereof) has been used as an informal benchmark for process modeling languages and BPMS.

We can characterize the workflow patterns into three main categories: i) patterns that are handled by CMMN basic constructs, ii) patterns that rely on CMMN's engine capabilities and iii) patterns that cannot be handled by current CMMN specification. A CMMN tool builder can propose patterns of the second category as companion modeling idioms, which can be translated behind the scenes into standard CMMN. The third category is problematic, however, since its support in CMMN tools will break model interoperability.

This paper is structured as follows. In the next chapters, we briefly introduce the Workflow Patterns and the CMMN standard. Chapter 4 divide the workflow patterns into the three aforementioned categories, explaining how CMMN supports each pattern and the differences between the pattern definitions and the way it is supported by CMMN. We present some related works in Chapter 5 and will conclude and discuss our future steps in Chapter 6.

[1]CMMN does allow some tasks to be specified *internally* in an imperative style, e.g. BPMN, but the overall *structure* of the case is declarative. See section "CMMN"

Chapter 2

Workflow Patterns

The Workflow Patterns Initiative proposed, in 1999, a set of workflow patterns. These workflow patterns describe process modeling requirements in an implementation independent manner and distill the essential features of many workflow management systems [4]. The goal is that the workflow patterns could (i) allow the unbiased comparison between techniques for modeling business processes, (ii) the refinement of the existent techniques, or (iii) support the development of new techniques.

Initially, the focus was given only to control flow, with an original set of 20 workflow patterns. Over time, other patterns integrated this set. Other perspectives were also added such as the resource [14], data [15], and exception patterns [13]. The control flow perspective describes tasks in their execution order through the definition of different constructors, which allow the control of the execution flow. The resource perspective provides a structural organization defining human and/or devices responsible for executing sub-processes. The data perspective creates a layer between the business and data processing, in which data objects or variables can affect pre- and post-conditions of a sub-process execution. And the exception handling perspective deals with undesirable events encountered during the execution. The workflow patterns initiative affirms it is only possible to specify handlers for expected types of exception. With this constraint in mind, they determined the range of exception events that can be detected and provide a useful basis for recovery handling.

The patterns are based on a detailed analysis of various BPM systems. Practical experiences obtained through a multitude of evaluations resulted in reformulations and refinements of the patterns. The Workflow Patterns Initiative claims that they looked at the frequencies of patterns in real-life projects [4].

Chapter 3

CMMN

In 2009, the Object Management Group (OMG) issued a request for proposal (RFP) for the creation of a standard modeling notation for use with case management. The result of the OMG's effort was CMMN 1.0 [12], which was published in 2014. CMMN is a data-centric approach based on business artifacts [8]. One of the main differences between CMMN and other languages as BPMN and EPC is the paradigm shift from prescriptive to declarative.

In a declarative model, it is not possible to predict a flow of tasks. Which tasks and the order they will execute is a decision to be taken by the knowledge worker. Knowledge workers are the experts in the domain who execute different instances of the model (called a *case* in the case management domain), e.g. a doctor or a nurse for the healthcare domain, or a judge for the law domain. Therefore, the knowledge worker is responsible for solving a case following his expertise and the constraints of the model.

A CMMN model primarily comprises the following items [12]:

- *case plan model*: captures the complete behavioral model of a case;
- *task*: defines an atomic unit of work. Four types of tasks are supported: *human* (performed by a knowledge worker), *process* (to embed a process, e.g. a BPMN model), *decision* (to embed a decision, e.g. a DMN model) and *case* (to embed other cases e.g. other CMMN models);
- *stage*: serves as a container of elements for case plans;
- *sentry*: "watches out" for important situations to occur (or events), which influence the further proceedings of a case;
- *event listener*: captures events, which are things that "happen" during a case. Events may trigger, for example, the enabling, activation, and termination of stages and tasks, or the achievement of milestones. The event listener can be specialized into: *timer event listener* and *user event listener*;

- *milestone*: represents an achievable target, defined to enable evaluation of progress of the case;
- *case file item*: represents a piece of information of any nature, ranging from unstructured to structured, and from simple to complex, and can be defined based on any information modeling language;
- *connectors*: serves to visualize complex dependencies between elements;
- *discretionary items*: identifies an item, of which instances can be planned, to the "discretion" of a case manager.

For a CMMN model, there exist two phases. During the design phase, the business analyst/modeler defines a set of tasks (predefined tasks for all cases), constraints (rules that must be respected during the execution of a case) and discretionary items (to be planned at runtime by the knowledge worker). In the runtime phase, the knowledge worker executes the model following the predefined plan, but choosing the set of tasks and their order to fit his needs. In addition, he can also "adjust" the case, planning the discretionary items to become concrete and to be included in the execution.

Chapter 4

Expressing Workflow Patterns in CMMN

In this work, we relied on a well established set of workflow patterns to evaluate the CMMN standard. In particular, we were interested to assess whether and to what extent we could implement the behaviors expressed by the workflow patterns in CMMN. Therefore, our analysis intends to investigate the extent to which a business analyst would be able to incorporate imperative modeling idioms while modeling declarative processes.

Before diving into the details of our classification, we would like to note that we tried to respect the spirit of case management while translating the workflow patterns into CMMN. Indeed, case management in general (and CMMN in particular) account for knowledge worker involvement, who is asked to react to the best of his/her knowledge/abilities to the context s/he faces in a particular context and adapt the business process as needed. Therefore, although a workflow pattern defines a behavior to be followed, in CMMN this behavior is only offered to the knowledge worker and not imposed. Thus, the CMMN processes we propose should cover the essence of the considered pattern but still leave the ability to the knowledge worker to adapt it. Consider for example the *exclusive choice pattern* giving multiple alternative courses, from which only one must be taken. In translating this pattern, we want to enforce that no more than one alternative could be followed by the knowledge worker but we will overlook the fact that the imperative pattern forces the choice to be made at a particular instant during process execution.

4.1 Workflow patterns supported by CMMN constructs

In this section, we will consider the workflow patterns that we were able to express solely by relying on CMMN core language constructs as defined in the CMMN specifications. We will address each of the workflow pattern concerns (control flow, data

management, etc.) in turn and will provide, for each, some examples of pattern translations into CMMN.

4.1.1 Control flow

CMMN supports the elementary aspects of process control and all basic control flow patterns can be represented. This set is composed of five patterns: *sequence*, *parallel split*, *synchronization*, *exclusive choice*, and *simple merge*. The *sequence* pattern establishes that one task can only be enabled after the completion of another task. The CMMN model in Figure 4.1 makes use of an *EntryCriterion Sentry* (represented as the shallow diamond shape) to enable *Task B* only after the completion of *Task A*.

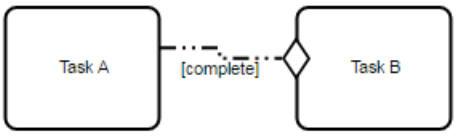
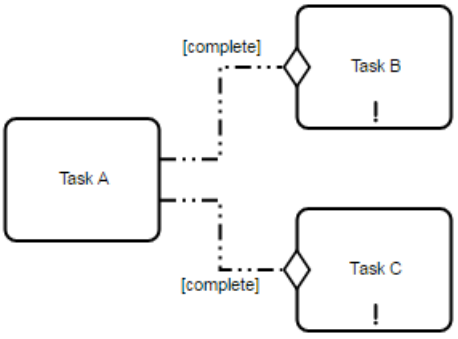
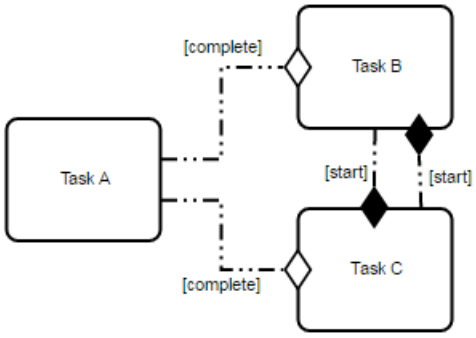


Figure 4.1: *Sequence* pattern in CMMN.

Split patterns introduce the divergence of an execution branch. The two split patterns in this category are the *parallel split* and the *exclusive choice*, shown in Figure 4.2(a) and 4.2(b) respectively. The former implies that all the execution of branches are executed, while the latter requires that only one among the subsequent branches is taken.



(a) Parallel Split



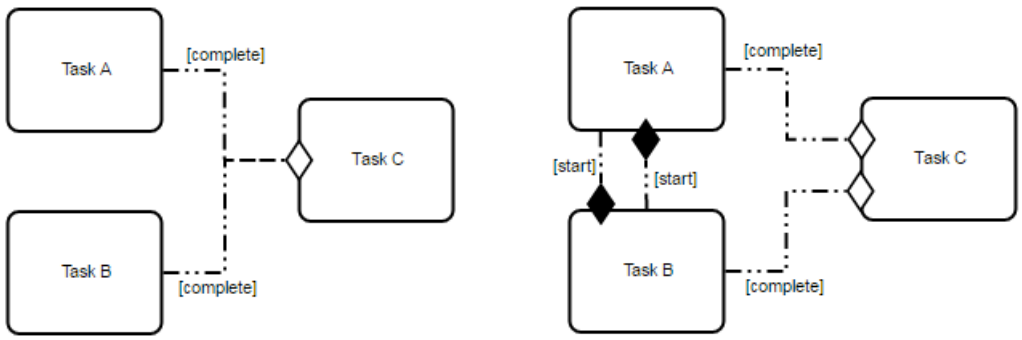
(b) Exclusive Choice

Figure 4.2: Basic split patterns in CMMN.

Figure 4.2(a) shows the CMMN model that represents the *parallel split*. In order to make explicit that all the branches need to be executed, we use the *required* decorator (depicted by an !). However, as we mentioned in Chapter 3 the knowledge worker always has the ability to skip one task (or both) and can even decide not to execute *Task B* or *C* right after *Task A*. We respected this behavior in the CMMN model.

The CMMN model shown in Figure 4.2(b) corresponds to the *exclusive choice* pattern. As for the *parallel split*, the model also enables all subsequent "branches". However, once one of them is chosen by the knowledge worker to be executed, all other branches are terminated and are not longer offered to him. This is achieved by the *ExitCriterion Sentry* (the solid diamond shape) that terminates a task when certain conditions hold. In this example, the sentry's condition is the start of the first task of another branch. For example, in Figure 4.2(b), starting *Task B* terminates *Task C* as expressed by *Task C*'s exit criterion.

On the other hand, merge patterns introduce the convergence of branches into one execution branch. The basic merge patterns are *synchronization* and *simple merge* (Figures 4.3(a) and 4.3(b) respectively). The first merge pattern requires the completion of all branches before enabling the subsequent branch. To model this behavior in CMMN, the availability of the task following the merge depends on the completion of all the branches. To do so, we rely on the *EntryCriterion Sentry* that prevents the availability of Task C before the completion of both Task A and Task B.



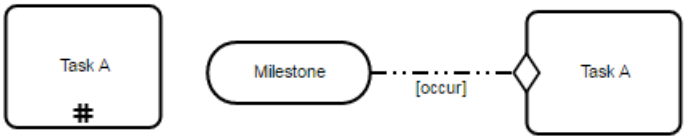
(a) Synchronization (b) Simple Merge

Figure 4.3: Basic merge patterns in CMMN.

The *simple merge* uses a different *EntryCriterion Sentry* for each branch. When using multiple sentries, only one of them must meet it's condition for the task to be enabled. To avoid the knowledge worker to execute more than one branch, we use *ExitCriterion Sentries* as they were used for the *exclusive choice*.

CMMN also offers a set of decorators and modifiers letting the modeler introduce some element-specific constraints or allowing specific behaviors on them. The *repetition* decorator, denoted by the # symbol, allows for a given element (task or stage) to be executed more than once (see Figure 4.4(a)). This behavior matches the *multiple instances without synchronization* workflow pattern. The *milestone*, is a CMMN construct that represents an achievable target, defined to enable evaluation of progress of a case. This corresponds to the control pattern also named *milestone* that requires the ability of enabling a task based on such achievable targets. Figure 4.4(b) shows an example of the use of a *milestone* in a CMMN model.

Some advanced branching and synchronizing patterns can be represented with



(a) Multi Instances Without Synchronization

(b) Milestone

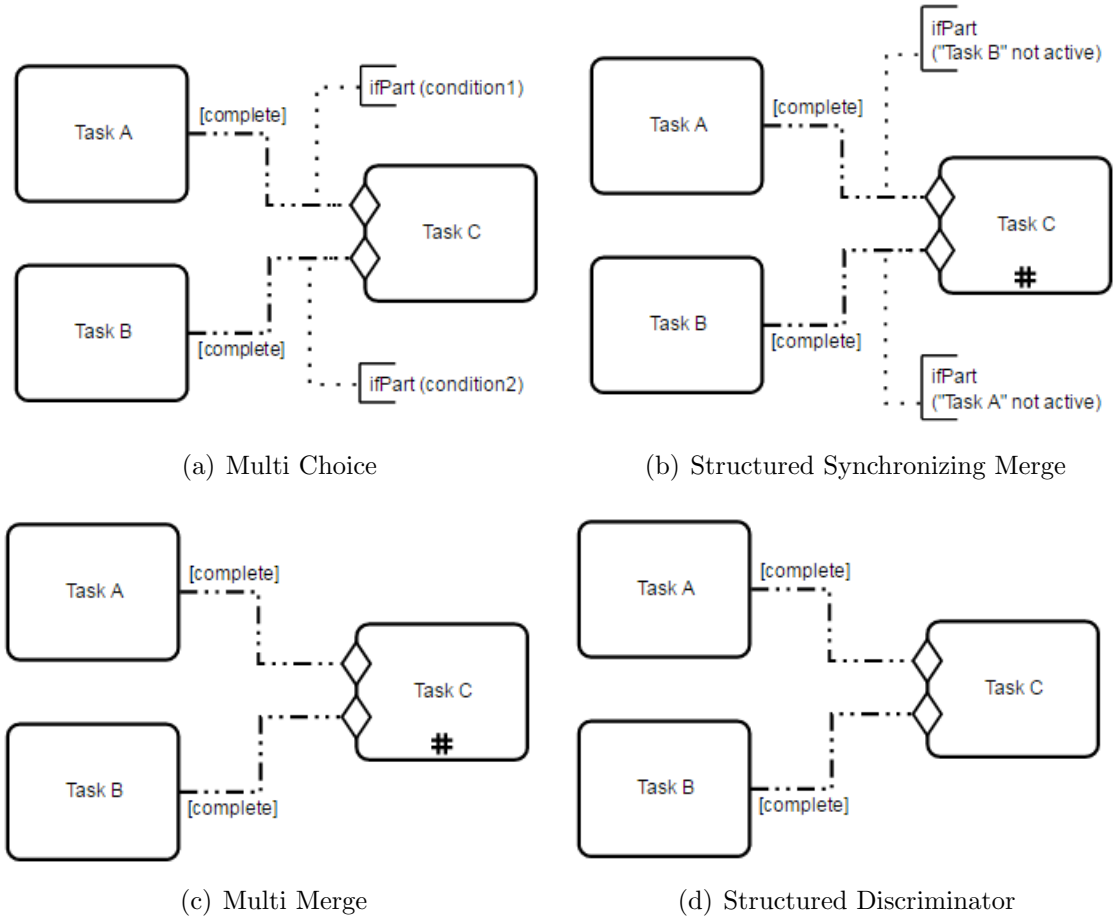
Figure 4.4: Element-specific constraints in CMMN.

little difference for the ones aforementioned, all of them shown in Figure 4.5. The *multi choice* pattern in Figure 4.5(a) uses the *ifPart* of the entry sentries to describe conditions that will determine which branch would be enabled. The *structured synchronizing merge* pattern in Figure 4.5(b) also uses the *ifPart* of entry sentries to control the enablement of *Task C* only after the execution of the active branches. Instead of the conditions, the *multi merge* pattern in Figure 4.5(c) adds a *repetition* decorator to *Task C*, so that each time one sentry is evaluated to true, one instance of *Task C* would be created. The *structured discriminator* pattern in Figure 4.5(d) does not need the decorator because *Task C* should be enabled only once, after the completion of one of the precedent tasks.

Two other advanced patterns can be represented in CMMN: *blocking discriminator* and *cancelling discriminator*. The *blocking discriminator* pattern determines that the subsequent task should be enabled with the completion of the one of the precedent tasks. The particularity is that subsequent enablement are blocked until the *blocking discriminator* construct resets. Figure 4.6(a) shows that *Tasks A* and *B* will be enabled when *Task C* is not yet executed, or after each completion of this task, and *Task C* will be enabled after the completion of *Task A* or *B* but not after the completion of the second one. The same behavior is defined for the *cancelling discriminator* pattern. The difference is that, instead of blocking subsequent enablements, once one task (*A* or *B*) is completed, the other one is canceled. Figure 4.6(b) shows this pattern.

Patterns part of the subset of *cancellation and force completion patterns* can also be expressed in CMMN, again relying on the *ExitCriterion Sentry* previously introduced. We have seen how a given task could be terminated by the completion of another task, which permits to model the *cancel task* pattern (see Figure 4.7(a)). Similarly, the *ExitCriterion Sentry* can be associated to enclosing elements as a given *Stage* or the whole CMMN case plan in order to express the *Cancel Region* (shown in Figure 4.7(c)) and the Cancel Case (shown in Figure 4.7(b)) workflow patterns, respectively.

In the termination category, we find two control patterns: the *explicit termination* and the *implicit termination*. The former establishes that when a certain end node is reached, any remaining work in the process instance is canceled and the overall process instance is recorded as having completed successfully, regardless of whether



(a) Multi Choice

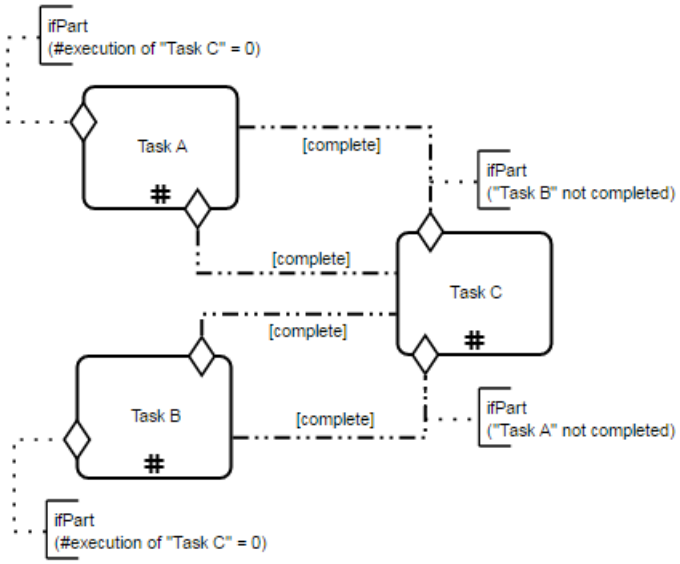
(b) Structured Synchronizing Merge

(c) Multi Merge

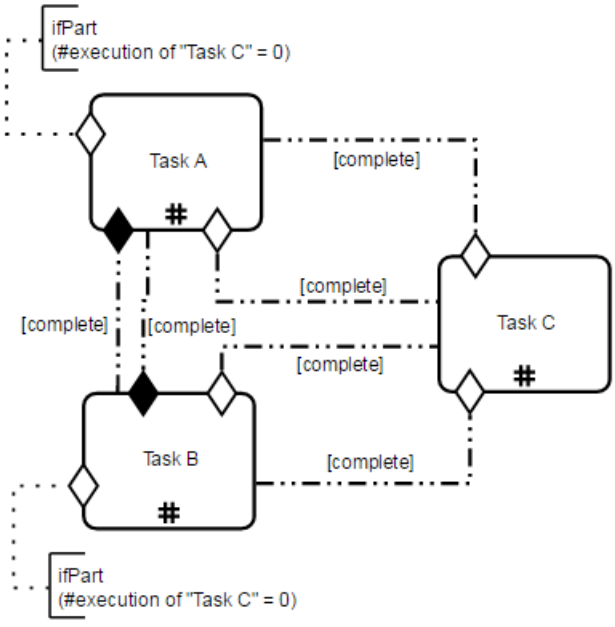
(d) Structured Discriminator

Figure 4.5: Advanced branching and synchronizing patterns in CMMN.

there are any tasks in progress or remaining to be executed. We can represent this behavior by the use of an *ExitCriterion Sentry* associated to the *case plan model*, illustrated in Figure 4.8. In the example, we considered *Task B* as the end node of the process. The ■ symbol associated to the *case plan model* denotes an *auto complete* decorator. The *auto complete* property determines that the related item will be moved automatically to the *completed* state when all of its children are either completed or not expected to run (i.e. in a *disabled*, *terminated*, or *failed* state) and all of the *required* tasks are *completed*. This behavior can be related to the *implicit termination* pattern instructing that a given process (or sub-process) instance should terminate when there are no remaining work items that could still be performed either now or at any time in the future and the process instance is not in deadlock. However, there are some behavioral differences that should be noted, due to the flexibility introduced by case management paradigm. Indeed, since a knowledge worker can willingly discard a non required task, the case might terminate, even though some tasks are still offered (the ones discarded). To mimic the exact behavior, a *required* decorator must be set on all the tasks and stages (see Task A, figure 4.8).

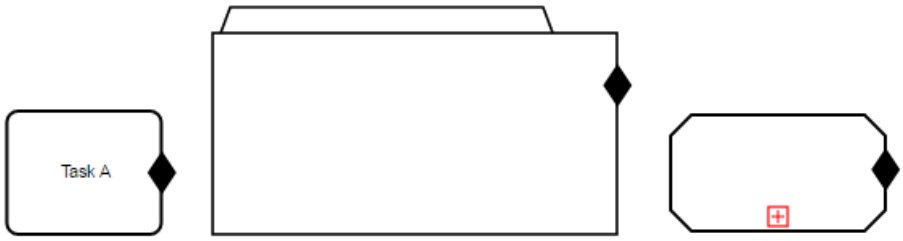


(a) Blocking Discriminator



(b) Cancelling Discriminator

Figure 4.6: Other advanced branching and synchronizing patterns in CMMN.



(a) Cancel Task (b) Cancel Case (c) Cancel Region

Figure 4.7: Cancellation patterns

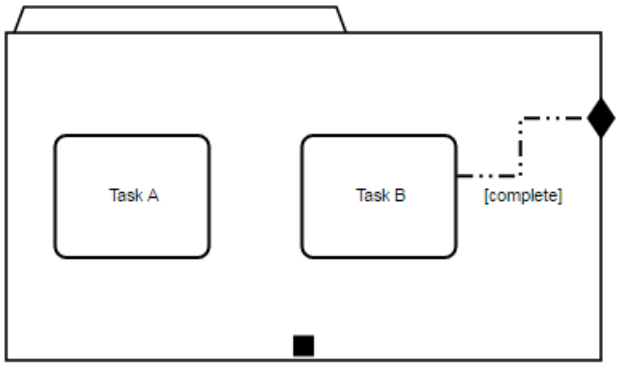


Figure 4.8: *Auto complete* and *ExitCriterion Sentry* in CMMN as termination.

Finally, multiple patterns are *de facto* supported by intrinsic characteristics of declarative process modeling languages (CMMN in particular) and, thus, do not require any specific construct. This is the case of the *deferred choice* pattern that enables a branch, depending on interaction with the operating environment. In contrast to the *exclusive choice* pattern, this pattern introduces a race condition between the different branches. This race condition is implicit in a case management process since all the tasks can be available simultaneously to many knowledge workers and might be activated by any event related to the environment. A similar reasoning can be applied to the *arbitrary cycles* and *structured loop* patterns, which introduce the ability to run a given task (or a set thereof) multiple times. Indeed, CMMN does not depend on a structured model and it is up to the knowledge worker to decide on the sequence s/he wants to repeat.

4.1.2 Data

To the extent that CMMN is considered as data-centric [7], it should come as no surprise that we were able to represent all the data patterns in CMMN. Data related aspects in a CMMN business process model relies on the *CaseFile* construct. It is a structure that serves as a container for data and as a context for raising events and evaluating expressions. The data (or data structure) inside a *CaseFile* can represent

data inputs and/or outputs of tasks.

A *CaseFile* is the set of all *CaseFile Items* in a case model. Figure 4.9 shows an example of *CaseFile Item* with some data. The figure shows a task having a condition related to a *CaseFile Item*. The condition is defined as an expression in the specification of the *EntrySenty*.

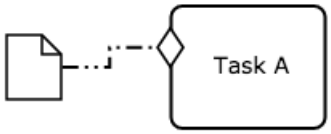


Figure 4.9: *CaseFile Items* in CMMN.

Using the *CaseFile Item* as a precondition to a task, it is possible to represent all of the *task precondition* patterns. For example, if we might verify the existence of some data in the *CaseFile Item*, that matches *Task Precondition - Data Existence* pattern. On the other hand, verifying the value of one parameter in the *CaseFile Item* will match the *Task Precondition - Data Value* pattern.

In CMMN, a *CaseFile Item* is designed with the Content Management Interoperability standard in mind which is a content management standard that allows interoperability between an engine and a content management system [3]. This interoperability allows the specification of the patterns in the external interaction category, depending on the way the parameters are updated and/or read. Figure 4.9 can represent an external or internal data interaction pattern. In this example, it represents a pattern that mentions a task as one of the elements related, e.g. the *Data Interaction - Environment to Task - Pull-Oriented* pattern. Notice that the same can be done when other elements are involved.

Regarding the Transfer patterns, they cannot be represented in CMMN as defined. We can represent similar behavior if one task updates some data in a *CaseFile Item* and another task reads the same data.

4.1.3 Resources

CMMN supports the definition of roles, which authorizes knowledge workers or teams of knowledge workers to perform human tasks, to plan based on discretionary items, and to raise user events. A role in CMMN can be a participant (e.g. each person in a department) or a set of participants (e.g. managers, or directors).

From this ability, two patterns can be represented: *direct distribution* and *role-based distribution*. The *direct distribution* pattern requires assigning one participant directly to the execution of a task. CMMN can do that when participants are defined as roles. The *role-based distribution* requires assigning teams of participants to the execution of a task. To do so, the roles in the model should be defined as teams of participants. Note, however, that CMMN supports neither the assignment of roles to participants (e.g. it is not possible to say that Paul is a manager) nor the definition of devices as resources.

4.1.4 Exception handling

Three of these patterns are supported in CMMN:

1. *work item failure*: characterizes the inability of the work item to progress any further. The life cycles in CMMN contain the *failed* state, which is a semi-final state and indicates an exception or software failure. This state can be used as any other state in CMMN, such as for enabling or terminating some task, for triggering an event or for achieving some milestone.

2. *deadline expiry*: determines the ability of specifying a deadline to indicate when the item should be completed, although deadlines for commencement are also possible. Such deadlines can be expressed in CMMN through the definition of *user event listeners*. Figure 4.10(a) shows an example of *user event listeners* (denoted by a double line circle with a clock symbol) as deadline for a task to be completed. If the task is not completed in the deadline, the task will be terminated. And Figure 4.10(b) shows an example where *Task A* will only be enabled after the deadline specified by the *user event listener*.

(a) (b)

Figure 4.10: *Deadline expiry* pattern in CMMN.

3. *external trigger*: signals the occurrence of an event that impacts the item and that requires some form of handling. This kind of signal, in CMMN, can be represented as a *user event listener*. In this case, the user (the knowledge worker) is responsible of signaling any external event that happens. The advantage here is that the knowledge worker is the expert in the domain to deal with the situation and to decide how to solve it.

4.2 Workflow patterns supported by CMMN engines' scripting capabilities

CMMN supports the definition of four types of *task decorators* that customize the basic behavior of a task. Such decorators are expressed in terms of different *behavioral rules*:

1. *applicability rule*: such rules are used exclusively for *discretionary items*. They determine whether tasks should be shown to the knowledge worker/case manager, and who can make that decision during runtime;

2. *manual activation rule*: defines whether an item needs a human intervention to be activated or not;

3. *required rule*: when associated to an item demands its execution (successfully or not, termination, or being manually disabled). While the item is not executed, its parent cannot transit to the *completed* state;

4. *repetition rule*: when associated to an item allows the control of the number of instances of this item should occur.

In CMMN, these rules are defined in terms of *expressions*, where an expression is composed of a *body*, which is the expression itself, and the *language* in which the expression is encoded. However, CMMN does not prescribe a specific language for such expressions, nor does it specify the required 'richness'/power of such languages. It is up to the CMMN engine's implementers.

Many of the workflow patterns in this category *can be* expressed using one of the four behavioral rules shown above, *provided that the CMMN engine supports a scripting language powerful enough to represent the required conditions*. This is the case for a number of workflow patterns from the *control*, *resources*, and *exception handling* categories, as illustrated below.

4.2.1 Control flow

Some patterns require dealing with multiple instances of the same task. We mentioned in section 4.1 that the *Multiple Instances Without Synchronization* pattern can be supported using CMMN's *repetition* decorator for tasks. Indeed, through the *repetition rule*, we can support the definition of parameters such as the number of instances to execute, the number of instances expected to complete, or the action to be applied to the instances in execution that exceed the expected number. Further, depending on the rule specification language supported by the engine, we can make it possible to specify the values of such parameters statically or dynamically, which would enable us to support all the patterns in the multiple instance category.

Some examples of patterns in the multiple instances category follows:

- the *Multiple Instances with a Priori Design-Time Knowledge* pattern requires a static number of instances to be executed, and all instances are expected to complete;

- the *Multiple Instances with a Priori Run-Time Knowledge* is the same pattern, but the number of instances is statically defined at runtime;

- the *Multiple Instances without a Priori Run-Time Knowledge* is also the same pattern, with the number of instances dynamically defined;

- the *Canceling Partial Join for Multiple Instances* pattern requires a static number of instances to execute, a static expected number of instances to be completed, and the remaining tasks should be canceled;

- the *Dynamic Partial Join for Multiple Instances* pattern requires dynamic number of instances, dynamic number of expected completions, and the remaining tasks are inconsequential.

4.2.2 Resources

Almost all resources patterns can be supported by engine's scripting capabilities. We see two ways to increase resources support by a CMMN engine:

1. through *applicability rules* which allows the assignment of roles to discretionary tasks. Such tasks are shown to be planned only to authorized roles. It is the engine's scripting capabilities who interpret *applicability rules* that limits the assignment of different roles, referring or not to other resources.

2. through simple extensions to the specification that allows the engine to remains compliant to the standard. For example, extending CMMN specification to support relation between roles, assignment of participant to teams, or different kind of resources (e.g., devices, or human resources).

4.2.3 Exception handling

One pattern in this category that could be support by the engine is the *resource unavailability*. The support of this pattern would be a complement of the support to the resource patterns. The engine could support the definition of actions to deal with problems happened in distribution time - no resource can be found which meets the specified distribution criteria - or happened at some time after allocation - the resource is no longer able to undertake or complete the work.

The other one is *constraint violation*. If the engine supports a monitoring mechanism, the engine could monitor when some constraint is violated by the knowledge worker and warn him/her. Notice that, in CMMN, the knowledge worker is who always decide what to do next, thus some violations could not be seen as a real violation. Although we believe it is important to notify the knowledge worker about the violation, and when possible, about the consequences to the rest of the execution.

4.3 Workflow patterns not supported by the current CMMN specification

There are behaviors not intrinsically supported and that cannot be reproduced using the current CMMN specification. Such behaviors mostly disrespect the declarative nature of CMMN and the knowledge worker decision-making characteristic. Unknown reasons and/or arguments can come up to justify changes in the specification, and so other versions can support such behaviors. In this section, we show some of these behaviors and patterns related to them.

4.3.1 Control flow

One behavior that disrespect the power of decision of the knowledge worker is when avoiding the concurrency between tasks. Following the CMMN spirit, the knowledge worker can always decide to execute tasks concurrently. However, workflow patterns as *critical section*, *interleaved routing*, *interleaved parallel routing* demand the specification of tasks that cannot occur concurrently and so they cannot be represented in CMMN. Suppose the *critical section* pattern, which determines that two regions are defined in the process, and, when a task of one section starts executing, no task from the other section can execute in parallel. One could think that the solution is to play with variables to identify which section is currently executing and enable tasks in the same section as well as disable tasks in another section. Indeed, this is not a solution for CMMN, since the life cycle of a task (and other items) does not contain any automatic transition from the *enabled* state to *disabled* or *suspended*. The transitions between these states are manual transitions, meaning that the knowledge worker has the ability to decide of disabling or suspending an enabled task, but this control could not be done automatically by an engine.

As a CMMN model is not structured into corresponding splits and merges, patterns as *local synchronizing merge* and *global synchronizing merge* cannot be represented. The same occur with the *thread split* and *thread merge* patterns, since we do not have only one thread of execution. The knowledge worker decides about the order to execute tasks and also about executing them in parallel or not.

4.3.2 Resources

The resource patterns in the auto-start category cannot be represented in CMMN. A task can be started in different ways: 1) manually, by the knowledge worker; 2) automatically, when the manual activation rule is evaluated to false; and 3) automatically, when an event is triggered. There is no way of a task to be started with the creation or allocation of a resource, for example.

Chapter 5

Related Work

There are several works that analyze the coverage that different languages provide for modeling the workflow pattern's common behavior:

Wohed et.al. [17] made an extensive analysis through all workflow patterns to identify which ones are supported, partially or not supported by BPMN. They conclude that BPMN supports the majority of the control flow patterns and nearly half of the data patterns, while the support for the resource perspective is minimal. In [2], Dumas et al. analyze the coverage that UML Activity diagrams (AD) provides to model workflow patterns. Although the authors affirm having evaluated the whole set of workflow patterns, only some of them are discussed in the paper.

Börger [1] faced the problem from a different perspective by defining Abstract State Machine (ASM) models for all 43 control flow patterns, which provide a "precise and truly abstract form" of each pattern. He also provided abstract models as a basis for an accurate analysis and evaluation of practically relevant control flow patterns. In particular, in connection with business processes and web services, preventing the pattern variety to grow without rational guideline. Jaeger et.al. [5] also used the set of control flow patterns to derive composition patterns for web services. Such composition patterns serve as a basis for aggregation of service properties regarding QoS dimensions. The resulting aggregation schema supports the same structural elements as found in workflows. In addition, the authors discuss the aggregation of different QoS dimensions.

Regarding the CMMN language, to the better of our knowledge, there is no research work which evaluates the CMMN standard in order to identify its coverage workflow patterns. However, other works evaluating CMMN based on other aspects can be found in the literature. Kurz et.al [7] examined the applicability of CMMN in terms of Adaptive Case Management (ACM) requirements. Although the authors showed that CMMN supports most ACM requirements, there are some ACM aspects that are not yet supported. Marin et.al. [9] used a Method Complexity approach to analyze the complexity of CMMN models, in terms of number of objects, relationships and properties, which allows the calculation of a cumulative complexity. This approach aims at modeling the same process with different languages to measure the complexity of the models. Based on their findings, CMMN holds much promise,

specially compared to BPMN.

Chapter 6

Conclusions

Workflow management systems are adept at managing processes with high-volume, low-variability, knowledge-poor. Imperative process modeling languages such as BPMN or EPC do a great job at capturing the semantics of such processes. However, such systems - and languages - do a poor job with *case management*, which involves (relatively) low-volume, high-variability, knowledge intensive processes, such as processing patients at an emergency room, or repairing airliner jet engines. CMMN was developed explicitly for such processes, by specifying what either *must be* done (e.g. checking a patient's blood oxygen level at triage) or what *must not* be done (e.g. performing X-rays more than twice in the same patient.), leaving the choice of what to do to the "case worker". Contrast this with imperative workflow languages that specify *all the possible execution paths* that a process (instance) can take. At first glance, declarative appear to be more powerful, and a legitimate question to ask is whether business analysts/modelers can use CMMN to model all of their processes. One way to test this hypothesis is to check whether - and how - the common workflow patterns of [14] can be handled in CMMN.

Unsurprisingly, the representation of the workflow patterns - which have a strong control component - was not straightforward. This was to be somewhat expected, given CMMN's declarative nature. We classified the workflow patterns in three categories:

1. *Patterns directly supported by CMMN constructs*: such patterns can be expressed in CMMN, respecting the letter and the spirit of the CMMN standard. Most control patterns, and all the data patterns fit into this category

2. *Patterns representable using CMMN engine's scripting capabilities*: CMMN standard includes few hooks to customize CMMN execution engines

3. *Patterns not supported by the current CMMN specification*: there are no CMMN compliant extensions that can handle such patterns. Examples include patterns that avoid concurrency between tasks.

That some patterns cannot be represented by CMMN should not come as a surprise: CMMN is *not procedural in nature*. Figure 6.1 shows the percentage of patterns

according to our categories. Note that CMMN allows for some tasks to be BPMN subprocesses: if some part of the case management at hand is systematic/procedural in nature, we can specify it using an imperative language such as a BPMN subprocesses. The summary of form which category each workflow pattern belongs can be found in Annexe A.

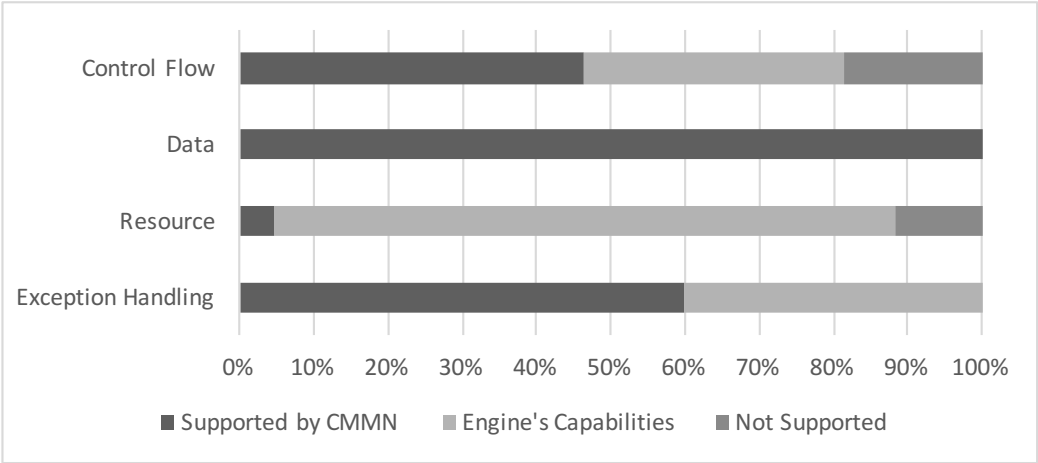


Figure 6.1: Percentage of workflow patterns per category.

What is more interesting is what we do with the first two categories. To the extent that such patterns are common and represent recurrent and commonly useful behavior, a CMMN modeling tool could offer them as *modeling idioms* to a business analyst (e.g. as part of the tool construct palette) that a modeler can select. A tool builder may even choose to use a custom notation for such patterns, as long as they are translated into standard CMMN behind the scenes. This will combine the advantages of modeling commodity and CMMN compliance, especially with regard to mode interchange. However, when such models are exchanged with other tools, we lose the 'pattern packaging'.

We believe there is a risk associated to the non-normative extensions proposed by specific engines. Indeed, while the standard explicitly states that the extensions should not modify the (standardized) behavior of a CMMN engine, there is no easy way to enforce compliance of the extensions, and it is left to the good will of the standard implementers. Consequently, the same process could behave differently in two different engines - thereby diminishing the value of model interchange between tools. For example, considering the repetition rule, two engines might interpret differently the same expression. Hence, two cases would have different behaviors even if the knowledge worker performs the same choices at runtime. Worse yet, an engine extension might not even be compliant with the standard, and we have no way (formal or practical) of checking that.

This is a preliminary study regarding the CMMN standard. For future steps, we think on developing a CMMN model checker to warn the user about unusual situations he could face at runtime. We also intend to implement and experiment the

use of a CMMN engine in a practical domain, as healthcare.

Annexe A

Table 6.1: Control Flow Patterns

Control Flow			
Pattern	Supported by CMMN	Engine capabilities	Not handled
Basic Control Flow Patterns			
1. Sequence	✓		
2. Parallel Split	✓		
3. Synchronization	✓		
4. Exclusive Choice	✓		
5. Simple Merge	✓		
Advanced Branching and Synchronization Patterns			
6. Multi Choice	✓		
7. Structured Synchronizing Merge	✓		
8. Multi-Merge	✓		
9. Structured Discriminator	✓		
28. Blocking Discriminator	✓		
29. Canceling Discriminator	✓		
30. Structured Partial Join		✓	
31. Blocking Partial Join		✓	
32. Canceling Partial Join		✓	
33. Generalized AND-Join		✓	
37. Local Synchronizing Merge			✓
38. General Synchronizing Merge			✓
41. Thread Merge			✓
42. Thread Split			✓
Multiple Instance Patterns			
12. Multiple Instances without Synchronization	✓		
13. Multiple Instances with a Priori Design-Time Knowledge		✓	
14. Multiple Instances with a Priori Run-Time Knowledge		✓	

Table 6.1: (continued)

Control Flow			
Pattern	Supported by CMMN	Engine capabilities	Not handled
15. Multiple Instances without a Priori Run-Time Knowledge		✓	
34. Static Partial Join for Multiple Instances		✓	
35. Canceling Partial Join for Multiple Instances		✓	
36. Dynamic Partial Join for Multiple Instances		✓	
State-Based Patterns			
16. Deferred Choice	✓		
17. Interleaved Parallel Routing			✓
18. Milestone	✓		
39. Critical Section			✓
40. Interleaved Routing			✓
Cancellation and Force Completion Patterns			
19. Cancel Task	✓		
20. Cancel Case	✓		
25. Cancel Region	✓		
26. Cancel Multiple Instance Activity		✓	
27. Complete Multiple Instance Activity		✓	
Iteration Patterns			
10. Arbitrary Cycles	✓		
21. Structured Loop	✓		
22. Recursion			✓
Termination Patterns			
11. Implicit Termination	✓		
43. Explicit Termination	✓		
Trigger Patterns			
23. Transient Trigger		✓	
24. Persistent Trigger		✓	

Table 6.2: Resource Patterns

Resource			
Pattern	Supported by CMMN	Engine capabilities	Not handled
Creation Patterns			
1. Direct Distribution	✓		
2. Role-Based Distribution	✓		
3. Deferred Distribution		✓	
4. Authorization		✓	
5. Separation of Duties		✓	
6. Case Handling		✓	
7. Retain Familiar		✓	
8. Capability-Based Distribution		✓	
9. History-Based Distribution		✓	
10. Organizational Distribution		✓	
11. Automatic Execution		✓	
Push Patterns			
12. Distribution by Offer - Single Resource		✓	
13. Distribution by Offer - Multiple Resources		✓	
14. Distribution by Allocation - Single Resource		✓	
15. Random Allocation		✓	
16. Round Robin Allocation		✓	
17. Shortest Queue		✓	
18. Early Distribution		✓	
19. Distribution on Enablement		✓	
20. Late Distribution		✓	
Pull Patterns			
21. Resource-Initiated Allocation		✓	
22. Resource-Initiated Execution - Allocated Work Item		✓	
23. Resource-Initiated Execution - Offered Work Item		✓	
24. System-Determined Work Queue Content		✓	
25. Resource-Determined Work Queue Content		✓	
26. Selection Autonomy		✓	
Detour Patterns			
27. Delegation		✓	

Table 6.2: (continued)

Resource			
Pattern	Supported by CMMN	Engine capabilities	Not handled
28. Escalation		✓	
29. Deallocation		✓	
30. Stateful Reallocation		✓	
31. Stateless Reallocation		✓	
32. Suspension-Resumption		✓	
33. Skip		✓	
34. Redo		✓	
35. Pre-Do		✓	
Auto-Start Patterns			
36. Commencement on Creation			✓
37. Commencement on Allocation			✓
38. Piled Execution			✓
39. Chained Execution			✓
Visibility Patterns			
40. Configurable Unallocated Work Item Visibility		✓	
41. Configurable Allocated Work Item Visibility		✓	
Multiple Resource Pattern			
42. Simultaneous Execution		✓	
43. Additional Resources		✓	

Table 6.3: Data Patterns

Data			
Pattern	Supported by CMMN	Engine capabilities	Not handled
Data Visibility			
1. Task Data	✓		
2. Block Data	✓		
3. Scope Data	✓		
4. Multiple Instance Data	✓		
5. Case Data	✓		
6. Folder Data	✓		
7. Workflow Data	✓		
8. Environment Data	✓		
Internal Data Interaction			
9. Data Interaction - Task to Task	✓		

Table 6.3: (continued)

Data			
Pattern	Supported by CMMN	Engine capabilities	Not handled
10. Data Interaction - Block Task to Sub-Workflow Decomposition	✓		
11. Data Interaction - Sub-Workflow Decomposition to Block Task	✓		
12. Data Interaction - to Multiple Instance Task	✓		
13. Data Interaction - from Multiple Instance Task	✓		
14. Data Interaction - Case to Case	✓		
External Data Interaction			
15. Data Interaction - Task to Environment - Push-Oriented	✓		
16. Data Interaction - Environment to Task - Pull-Oriented	✓		
17. Data Interaction - Environment to Task - Push-Oriented	✓		
18. Data Interaction - Task to Environment - Pull-Oriented	✓		
19. Data Interaction - Case to Environment - Push-Oriented	✓		
20. Data Interaction - Environment to Case - Pull-Oriented	✓		
21. Data Interaction - Environment to Case - Push-Oriented	✓		
22. Data Interaction - Case to Environment - Pull-Oriented	✓		
23. Data Interaction - Workflow to Environment - Push-Oriented	✓		
24. Data Interaction - Environment to Workflow - Pull-Oriented	✓		
25. Data Interaction - Environment to Workflow - Push-Oriented	✓		
26. Data Interaction - Workflow to Environment - Pull-Oriented	✓		
Data Transfer Patterns			

Table 6.3: (continued)

Data			
Pattern	Supported by CMMN	Engine capabilities	Not handled
27. Data Transfer by Value - Incoming	✓		
28. Data Transfer by Value - Outgoing	✓		
29. Data Transfer - Copy In/Copy Out	✓		
30. Data Transfer by Reference - Unlocked	✓		
31. Data Transfer by Reference - With Lock	✓		
32. Data Transformation - Input	✓		
33. Data Transformation - Output	✓		
Data-Based Routing			
34. Task Precondition - Data Existence	✓		
35. Task Precondition - Data Value	✓		
36. Task Postcondition - Data Existence	✓		
37. Task postcondition - Data Value	✓		
38. Event-based Task Trigger	✓		
39. Data-based Task Trigger	✓		
40. Data-based Routing	✓		

Table 6.4: Exception Handling Patterns

Exception Handling			
Pattern	Supported by CMMN	Engine capabilities	Not handled
1. Work Item Failure	✓		
2. Deadline Expiry	✓		
3. Resource Unavailability		✓	
4. External Trigger	✓		
5. Constraint Violation		✓	

Bibliography

[1] E. Börger, *Conceptual Modeling - ER 2007: 26th International Conference on Conceptual Modeling, Auckland, New Zealand, November 5-9, 2007. Proceedings*, ch. Modeling Workflow Patterns from First Principles, pp. 1–20. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.

[2] M. Dumas and A. H. M. Hofstede, "Uml activity diagrams as a workflow specification language," in *Proceedings of 4th International Conference on Modeling Languages, Concepts and Tools*, (Berlin, Heidelberg), pp. 76–90, Springer, 2001.

[3] A. O. S. for the Information Society (OASIS), "Content management interoperability services (cmis) - version 1.1," 2011.

[4] A. Hofstede, W. van der Aalst, M. Adams, and N. Russell, *Modern Business Process Automation: YAWL and Its Support Environment*. Springer Publishing Company, Incorporated, 1st ed., 2009.

[5] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl, "Qos aggregation for web service composition using workflow patterns," in *Proceedings of 8th Enterprise Distributed Object Computing Conference*, pp. 149–159, Sept 2004.

[6] M. Kurz and A. Fleischmann, *Proceedings of Second International Conference, Karlsruhe, Germany*, ch. BPM 2.0: Business Process Management Meets Empowerment, pp. 54–83. Berlin, Heidelberg: Springer, 2011.

[7] M. Kurz, W. Schmidt, A. Fleischmann, and M. Lederer, "Leveraging cmmn for acm: Examining the applicability of a new omg standard for adaptive case management," in *Proceedings of the 7th International Conference on Subject-Oriented Business Process Management*, (New York, NY, USA), pp. 4:1–4:9, ACM, 2015.

[8] M. Marin, R. Hull, and R. Vaculn, "Data centric bpm and the emerging case management standard: A short survey," in *Business Process Management Workshops*, vol. 132 of *LNBIP*, pp. 24–30, Springer Berlin Heidelberg, 2013.

[9] M. A. Marin, H. Lotriet, and J. A. Van Der Poll, "Measuring method complexity of the case management modeling and notation (cmmn)," in *Proceedings of the Southern African Institute for Computer Scientist and Information Technologists*

Annual Conference 2014 on SAICSIT 2014 Empowered by Technology, SAICSIT '14, (New York, NY, USA), pp. 209:209–209:216, ACM, 2014.

[10] H. Mili, G. Tremblay, G. B. Jaoude, É. Lefebvre, L. Elabed, and G. E. Boussaidi, "Business Process Modeling Languages: Sorting Through the Alphabet Soup," *ACM Computing Surveys*, vol. 43, no. 1, pp. 1–56, 2010.

[11] H. R. Motahari-Nezhad and K. D. Swenson, "Adaptive Case Management: Overview and Research Challenges," in *2013 IEEE 15th Conference on Business Informatics*, pp. 264–269, IEEE, 2013.

[12] O. M. G. (OMG), "Case management model and notation (cmmn) version 1.0," 2014.

[13] N. Russell, W. Aalst, and A. Hofstede, *Advanced Information Systems Engineering: 18th International Conference, CAiSE 2006, Luxembourg*, ch. Workflow Exception Patterns, pp. 288–302. Berlin, Heidelberg: Springer, 2006.

[14] N. Russell, W. M. P. Aalst, A. H. M. Hofstede, and D. Edmond, *Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005. Proceedings*, ch. Workflow Resource Patterns: Identification, Representation and Tool Support, pp. 216–232. Berlin, Heidelberg: Springer, 2005.

[15] N. Russell, A. H. M. Hofstede, D. Edmond, and W. M. P. der Aalst, *Proceedings of 24th International Conference on Conceptual Modeling, Klagenfurt, Austria*, ch. Workflow Data Patterns: Identification, Representation and Tool Support, pp. 353–368. Berlin, Heidelberg: Springer, 2005.

[16] M. Weske, *Business Process Management Concepts, Languages, Architectures*. Springer, 2012.

[17] P. Wohed, W. M. van der Aalst, M. Dumas, A. H. M. ter Hofstede, and N. Russell, "Pattern-based analysis of bpmn - an extensive evaluation of the control-flow, the data and the resource perspectives," tech. rep., BPMcenter.org, 2006.

[18] H. Wolf, K. Herrmann, and K. Rothermel, "Dealing with uncertainty," *ACM Transactions on Intelligent Systems and Technology*, vol. 4, pp. 1–23, sep 2013.